# 并查集及应用

## 一：并查集

是一种数据结构。主要解决元素分组问题，无向图有无环问题。管理一系列不相交的集合，支持2种操作，查找根节点：FindRoot，合并节点：MergeElemet.。

需要抽像的能力将问题抽象成图的动态连通性问题。

# 二：模板伪代码

## 最原始的

查：

int FindRoot(int\* parent, int ele)

{

int i;

if (ele != parent[ele]) {

return FindRoot(parent, parent[ele]);

}

return ele;

}

并：

void Merge(int\* parent, int x, int y)

{

int xRoot, yRoot;

xRoot = FindRoot(parent, x);

yRoot = FindRoot(parent, y);

parent[xRoot] = yRoot;

}

## 优化的

/\*路径压缩：查找根结点的时候节点将同一个集合的点的父节点都置为根节点：递归方法实现\*/其实也可以不用记忆，单纯找根节点就可以了

int FindRootCompress(int\* parent, int ele)

{

    if (ele != parent[ele]) {

        parent[ele] = FindRootCompress(parent, parent[ele]);

        return parent[ele];

    }

    return ele;

}

/\*

按秩合并：即将树层次低的往层次高树合，同高度的合并，父节点的秩要加1

\*/

void MergeEleByRank(int\* parent, int\* rank, int x, int y)

{

    int xRoot, yRoot;

    xRoot = FindRootCompress(parent, x);

    yRoot = FindRootCompress(parent, y);

    // 同属于一个集合无需合并

    if (xRoot == yRoot) {

        return ;

    }

    if (rank[xRoot] > rank[yRoot]) {

        parent[yRoot] = xRoot;

    }

    else if (rank[yRoot] > rank[xRoot]){

        parent[xRoot] = yRoot;

    }

    else {

        parent[xRoot] = yRoot;

        rank[yRoot]++;

    }

}

## 3.小实例

int main()

{

    int parent[Vertices] = {0};

    int rank[Vertices] = {0};

int i;

// 6对亲戚关系，先维护并查集

    int relations[][2] = {

        {0,1},

        {1,2},

        {2,5},

        {1,6},

        {3,6},

        {6,4},

    };

    for (i = 0; i < Vertices; i++) {

        parent[i] = i;

        rank[i] = 1;

    }

    for (i = 0; i < sizeof(relations) / sizeof(relations[0]); i++) {

        MergeEleByRank(parent, rank, relations[i][0], relations[i][1]);

}

// 查看6,9是不是亲戚

int 6Root = FindRootCompress(parent, 6);

int 9Root = FindRootCompress(parent, 9);

if (6Root != 9Root) {

printf(“6 and 9 is not relative\n”);

}

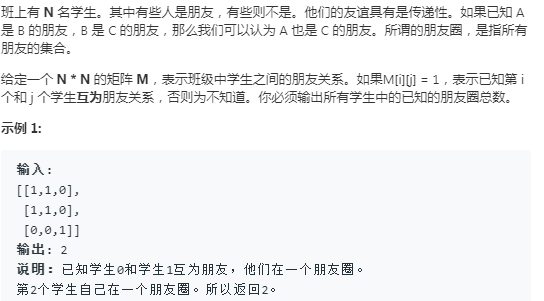
    printf("ele %d father is %d\n", 4, res);

    return 0;

}

# 三：应用及实例

## [547. 朋友圈](https://leetcode-cn.com/problems/friend-circles/)【并查集】



思路：

并查集做法：

初始朋友圈数为Msize（自己和自己算一个圈g\_cycle = MSize）,能merge的圈就少一个(g\_cycke--)，根据输入数组维护并查集，维护好了之后，圈就算出来了。

#include<stdio.h>

#include<stdlib.h>

/\*

并查集做法： 初始朋友圈数为Msize（自己和自己算一个圈g\_cycle = MSize）,能merge的圈就少一个(g\_cycke--)，根据输入数组维护并查集，维护好了之后，圈就算出来了。

对于数组元素为1的才需要去merge，且下表i、j不相同才去merge

\*/

int g\_cycle = 0;

static int box = 0;

int FindRootCompress(int\* parent, int x)

{

    box++;

    printf("x %d box %d\n",x, box);

    if (parent[x] == x) {

        return x;

    }

    parent[x] = FindRootCompress(parent, parent[x]);

    return parent[x];

}

int MergeEleByRank(int\* parent ,int\* rank, int x, int y)

{

    int xRoot, yRoot;

    xRoot = FindRootCompress(parent, x);

    yRoot = FindRootCompress(parent, y);

    if (xRoot == yRoot) {

        return 0;

    }

    g\_cycle--;

    if (rank[xRoot] > rank[yRoot]) {

        parent[yRoot] = xRoot;

    }

    else if(rank[yRoot] < rank[xRoot]) {

        parent[xRoot] = yRoot;

    }

    else {

        parent[xRoot] = yRoot;

        rank[xRoot]++;

    }

    return 1;

}

int findCircleNum(int\*\* M, int MSize, int\* MColSize)

{

    int\* parent = NULL;

    int\* rank = NULL;

    int i, j, iRoot, jRoot;

    int max = (MSize < MColSize[0]) ? MColSize[0] : MSize;

    parent = (int\*)calloc(max, sizeof(int));

    rank = (int\*)calloc(max, sizeof(int));

    g\_cycle = MSize;

    // 初始化，因为和自己都是在同一个朋友圈

    for(i = 0; i < max; i++) {

        parent[i] = i;

        rank[i] = 1;

    }

    for (i = 0; i < MSize; i++) {

        for (j = i + 1; j < MColSize[0]; j++) {

            if (M[i][j] != 1) {

                continue;

            }

            MergeEleByRank(parent, rank, i, j);

        }

    }

    return g\_cycle;

}

int main()

{

    int grid[][3] = {

        {1,1,0},

        {1,1,1},

        {0,1,1},

    };

    int i;

    int res = 0;

    int colSize = sizeof(grid[0]) / sizeof(grid[0][0]);

    int\*\* M = (int\*\*)calloc(sizeof(grid) / sizeof(grid[0]), sizeof(int\*));

    for (i = 0; i < sizeof(grid) / sizeof(grid[0]); i++) {

        M[i] = (int\*)calloc(sizeof(grid[0]) / sizeof(grid[0][0]), sizeof(int));

        memcpy(M[i], grid[i], sizeof(grid[0]) / sizeof(grid[0][0]) \* sizeof(int));

    }

    res = findCircleNum(M, sizeof(grid) / sizeof(grid[0]), &colSize);

    printf("%d\n", res);

}

## 2. [959. 由斜杠划分区域](https://leetcode-cn.com/problems/regions-cut-by-slashes/)【并查集+抽像转化】

## 

思路：

并查集思想，将其转化为连通问题。

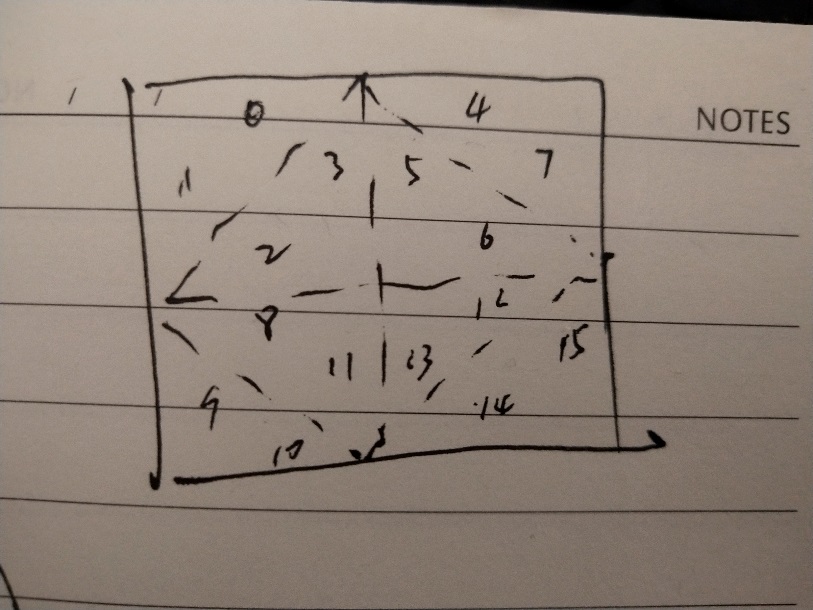
将每个小方格（grid由n\*n的小方格，小方格再转化为4个小三角形）。

从上方的0开始逆时针方向编号。后边的4、5、6、7（其实是0，1，2，3分别加4得到）。（1）对于一个小方格，如果碰到‘/’,需要0和1合并，以及2，3合并。如果碰到‘\’则合并0，3；以及合并1和2；如果碰到空，则合并1，2，3，4。

（2）除了自身合并，还要考虑相领的情形。

不管碰到设么符号，左边方格的3都是和右边方格的1（如图中的5）是连在一块的；上边的方格的2都是和下边方格的0（如图上的8是连载一块的），因此除了合并本方格的小三角还要考虑邻近的格子。

（3）处理完成之后，遍历parent数组，找parent[i] == I 的数量即集合数即可。

。

代码如下：

#include<stdio.h>

#include<stdlib.h>

int FindRootCompress(int\* parent, int ele)

{

    if (ele == parent[ele]) {

        return ele;

    }

    parent[ele] = FindRootCompress(parent, parent[ele]);

    return parent[ele];

}

int MergeElemetByRank(int\* parent, int\* rank, int x, int y)

{

    int xRoot, yRoot;

    xRoot = FindRootCompress(parent, x);

    yRoot = FindRootCompress(parent, y);

    if (xRoot == yRoot) {

        return 0;

    }

    if (rank[xRoot] > rank[yRoot]) {

        parent[yRoot] = xRoot;

    }

    if (rank[xRoot] < rank[yRoot]) {

        parent[xRoot] = yRoot;

    }

    if (rank[xRoot] == rank[yRoot]) {

        parent[xRoot] = yRoot;

        rank[yRoot]++;

    }

    return 1;

}

int MergeElemetByRank(int\* parent, int\* rank, int x, int y)

{

    int xRoot, yRoot;

    xRoot = FindRootCompress(parent, x);

    yRoot = FindRootCompress(parent, y);

    if (xRoot == yRoot) {

        return 0;

    }

    if (rank[xRoot] > rank[yRoot]) {

        parent[yRoot] = xRoot;

    }

    if (rank[xRoot] < rank[yRoot]) {

        parent[xRoot] = yRoot;

    }

    if (rank[xRoot] == rank[yRoot]) {

        parent[xRoot] = yRoot;

        rank[yRoot]++;

    }

    return 1;

}

int regionsBySlashes(char \*\* grid, int gridSize)

{

    // 将每个小格子都用'/'或'\'分为4个小三角形,从上放逆时针数为0，1，2，3

    int\* parent = NULL;

    int\* rank = NULL;

    int i, j = 0, col ,temp;

    int ans = 0;

    parent = (int\*)calloc(gridSize \* gridSize \* 4, sizeof(int));

    rank = (int\*)calloc(gridSize \* gridSize \* 4, sizeof(int));

    for (i = 0; i < gridSize \* gridSize \* 4; i++) {

        parent[i] = i;

        rank[i] = 1;

    }

    // 开始处理网络数组

    for (i = 0; i < gridSize; i++) {

        for (j = 0; j < gridSize; j++) {

            // 处理起始序号

            temp = i \* gridSize \* 4 + 4 \* j;

            // 合并0，1；合并2，3

            if (grid[i][j] == '/') {

                MergeElemetByRank(parent, rank, temp + 0, temp + 1);

                MergeElemetByRank(parent, rank, temp + 2, temp + 3);

            }

            // 合并0，2；合并1，3

            if (grid[i][j] == '\\') {

                MergeElemetByRank(parent, rank, temp + 0, temp + 3);

                MergeElemetByRank(parent, rank, temp + 1, temp + 2);

            }

            // 空格合并0，1，2，4

            if (grid[i][j] == ' ') {

                MergeElemetByRank(parent, rank, temp + 0, temp + 1);

                MergeElemetByRank(parent, rank, temp + 2, temp + 3);

                MergeElemetByRank(parent, rank, temp + 1, temp + 2);

            }

            // 如果不是第一行，向上合并，即本格的0和上边的2和并

            if (i != 0) {

                MergeElemetByRank(parent, rank, temp + 0, temp - 4 \* gridSize + 2);

            }

            // 不是最后一行的话，向下合并，即本格的2和下面的0

            if (i != gridSize - 1) {

                MergeElemetByRank(parent, rank, temp + 2, temp + 4 \* gridSize + 0);

            }

            // 不是第一列的话，向左合并，1和3

            if (j != 0) {

                MergeElemetByRank(parent, rank, temp + 1, temp - 4 + 3);

            }

            // 不是最后一列的话，向右合并

            if (j != gridSize - 1) {

                MergeElemetByRank(parent, rank, temp + 3, temp + 4 + 1);

            }

        }

    }

    for (i = 0; i < gridSize \* gridSize \* 4; i++) {

            if (parent[i] == i) {

                ans++;

            }

    }

    free(parent);

    free(rank);

    return ans;

}

int main()

{

    char\* tmpStr[] = {" /", "/ "};

    int i;

    int ret = 0;

    char\*\* grid = (char\*\*)calloc(sizeof(tmpStr) / sizeof(char\*), sizeof(char\*));

    for (i = 0; i < sizeof(tmpStr) / sizeof(char\*); i++) {

        grid[i] = (char\*)calloc(strlen(tmpStr) + 1, sizeof(char));

        memcpy(grid[i], tmpStr[i], strlen(tmpStr) + 1);

    }

    ret = regionsBySlashes(grid, 2);

    printf("ret %d\n", ret);

}